Tampereen ammattikorkeakoulu

Koneen häiriöajan seurantatyökalun toteutus

React Native -mobiilisovellus

Jesse Rönkkö

OPINNÄYTETYÖ
Toukokuu 2020

Tieto- ja viestintätekniikan koulutusohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikan koulutusohjelma
Ohjelmistotekniikka

RÖNKKÖ, JESSE:
Koneen häiriöajan seurantatyökalun toteutus
React Native -mobiilisovellus

Opinnäytetyö 36 sivua
Toukokuu 2020

Tässä opinnäytetyössä esitellään projekti, jossa tuotettiin toimiva työkalu logistiikkateollisuudessa käytettävien koneiden häiriöajan seurantaan. Työkalun tarkoituksena on kartoittaa tilanteita, ajankohtia ja sijainteja, joissa esiintyy koneen käyttäjän aiheuttamaa tai hänestä riippumatonta häiriöaikaa. Tässä opinnäytetyössä koneilla tarkoitetaan ihmisten operoimia liikkumiskykyisiä koneita, kuten trukkeja, ja häiriöajalla työaikaa, jolloin ehjää ja toimivaa konetta ei käytetä aktiivisesti työntekoon. Työn toimeksiantaja oli Cargotec Finland Oy.

Työkalu on Android-käyttöjärjestelmäisille mobiililaitteille React Native -sovelluskehyksellä kehitetty sovellus. Sovellus tarkkailee mobiililaitteen oman GPS:n, Wi-Fi:n ja mobiiliverkon avulla mobiililaitteen liikkumisnopeutta. Mobiililaite, jossa kehitettyä sovellusta käytetään, asetetaan tarkkailtavana olevaan koneeseen, joten koneen liikkumisnopeus on sama kuin kyydissä olevan mobiililaitteen. Liikkumisnopeuden perusteella sovellus määrittelee, onko kone tuotanto- vai häiriötilassa. Häiriötilan tapahtuessa sovellus pyytä käyttäjältä syytä häiriöajalle. Häiriötilan kesto, tapahtumasijainti sekä käytetyn mobiililaitteen ja koneen tunnus lähetetään pilveen. Kerättyä tietoa pystytään myöhemmin analysoimaan ja visualisoimaan esimerkiksi Amazon QuickSight -verkkotyökalulla häiriöaikaa aiheuttavien tilanteiden kartoittamiseksi.

Projektissa toteutetun työkalun on määrä päätyä mahdollisen asiakkaan koekäyttöön, mutta opinnäytetyön kirjoitushetkellä tätä koekäyttöä ei vielä ehditty tehdä. Työkalua koekäytettiin kuitenkin työkalun kehitystiimillä sekä toimeksiantajayrityksen sisäisellä asiakkaalla, ja se todettiin toimivaksi.

Tämä opinnäytetyö tuo ilmi React Nativen toimivuuden mobiilisovelluskehityksessä, jossa tarvittiin mobiililaitteen natiiveja rajapintoja, kuten sijainnin seurantaa ja muistinhallintaa sekä taustatilassa toimimista. Opinnäytetyössä todetaan myös React Nativessa toimivien uusien ominaisuuksien, hookkien ja Kontekstitilanhallintarajapinnan, kätevyys. Uusia ominaisuuksia verrataan niitä vastaaviin vanhempiin vaihtoehtoihin.

Asiasanat: android, mobiilisovellus, react native, sijainnin seuranta, hookit, konteksti-tilanhallintarajapinta

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
ICT Engineering
Software Engineering

RÖNKKÖ, JESSE:
Machine Downtime Tracking Tool
React Native Mobile Application

Bachelor's thesis 36 pages
May 2020

This thesis presents a project that produced a working tool for tracking the downtime of machines used in the logistics industry. The purpose of the tool is to monitor situations, times of day, and locations where there is a downtime caused by or independent of the person operating the machine. In this thesis, machines refer to human-operated mobile machines, such as forklifts, and downtime refers to the working time during which a working and operable machine is not in operation. The project was commissioned by Cargotec Finland Oy.

A mobile application was developed with the React Native application framework for Android mobile devices. The application monitors the movement speed of a mobile device using the device's GPS, Wi-Fi and mobile network. The mobile device is placed on the machine being observed to monitor its movement speed to determine whether the machine is in production or downtime state. When a downtime occurs, the application prompts the user to give a reason for the downtime. The duration of the downtime, the location of the event, and the identification number of the mobile device and the machine are logged and saved to the cloud. The collected data can later be analysed and visualised, for example, with the Amazon QuickSight web tool to detect the situations that cause downtime.

The tool implemented in the project is to end up in a trial run for a potential customer, but at the time of writing this thesis, the trial run had not yet been done. However, the tool was tested by the tool's development team and by the commissioner's internal customer, and the tool was found to be working as intended.

This thesis demonstrates the functionality of React Native in mobile application development, which requires the mobile device's native interfaces, such as location tracking, memory management, and background operation. The thesis also discusses the new features in React Native, the hooks, and the Context state management interface.

Key words: android, mobile application, react native, location tracking, hooks, context state management interface

SISÄLLYS

LYHENTEET JA TERMIT

frontend	Verkkopalveluiden ja sovellusten loppukäyttäjälle näkyvä osa.
backend	Verkkopalveluiden ja sovellusten tietokannan ja palvelimien ylläpidon käsittelevä osa.
GPS	Global Positioning System on satelliittipohjainen radionavigointijärjestelmä, jonka avulla voidaan määrittää sijainti, nopeus ja aika.
Wi-Fi	Langaton verkkoyhteysteknologia, joka tuottaa nopeita langattomia verkkoyhteyksiä radioaaltojen avulla.
rajapinta	Tietotekniikassa rajapinta mahdollistaa jonkin toiminnon integroimisen toiseen kokonaisuuteen.
natiivisovellus	Tietylle alustalle tehty sovellus, joka pääsee käsiksi alustaa käyttävän laitteen rajapintoihin.
natiivimoduuli	React Nativessa tapa luoda laitteen natiivilla koodilla JavaScriptille toimiva itsenäinen osa, joka käyttää laitteen natiivin puolen rajapintaa.
ohjelmistokehys	Valmiiksi tehty runko jonkin ohjelmistotuotteen kehitykselle.
ohjelmistokirjasto	Jonkin tietyn toiminnon toteuttava ohjelmistokoodista koostuva paketti.

1 JOHDANTO

Yritysmaailmassa tuotannon tehokkuuden maksimointi on tärkeää. Oleellinen osa tuotannon tehostamiselle on työajan käyttäminen mahdollisimman tehokkaasti ja joutoajan vähentäminen mahdollisimman pieneksi. Tämä opinnäytetyö koskee projektia, jossa kehitetyn mobiilisovelluksen on määrä toimia työkaluna kartoittamaan tilanteita, ajankohtia ja sijainteja, joissa esiintyy logistiikkateollisuudessa käytettävän koneen käyttäjän aiheuttamaa tai hänestä riippumatonta häiriöaikaa. Koneilla tässä opinnäytetyössä tarkoitetaan ihmisten operoimia liikkumiskykyisiä koneita, kuten trukkeja ja häiriöajalla tarkoitetaan työaikaa, jolloin ehjää ja toimivaa konetta ei käytetä aktiivisesti työntekoon.

Projektissa kehitetty sovellus julkaistiin Googlen omistamassa Google Play -nimisessä digitaalisessa sisältöpalvelussa. Sovellusta käyttävän yrityksen työntekijä voi ladata sovelluksen Android-käyttöjärjestelmäiselle mobiililaitteelle ja kirjautua hänelle määritetyin käyttäjätiedoin, sekä valitsemaan koneen yhtiönsä kirjaamista koneista.

Sovellusta käyttävä yritys asettaa pilveen sovellukselle omat asetuksensa ja ehtonsa omien tarpeidensa mukaan. Asetusarvoja ovat lista häiriöajan syille, vähimmäistuotantonopeus, häiriötilan aloittamisen kynnysaika ja häiriöajan syyn ilmoituksen kynnysaika. Vähimmäistuotantonopeus on liikkumisnopeus, jolla koneen katsotaan vielä olevan työnteossa eli tuotantotilassa. Häiriötilan aloittamisen kynnysaika on aikamäärä, jonka ajan koneen liikkumisnopeus voi olla hetkellisesti alle vähimmäistuotantonopeuden ennen kuin koneen katsotaan olevan häiriötilassa. Ilman häiriötilan aloittamisen kynnysaikaa koneen pikainen jarrutuskin saattaisi aloittaa häiriötilan ja häiriöajan laskemisen. Häiriöajan syyn ilmoituskynnysaika on aikamäärä, jonka aikana käyttäjä voi vielä ilmoittaa syyn häiriöajalle, vaikka kone olisi jo siirtynyt takaisin työntekoon eli pois häiriötilasta.

Sovellus ei tarvitse tietoa koneen antureista. Se määrittelee koneen liikkumisnopeuden mobiililaitteen oman GPS:n, Wi-Fi:n ja mobiiliverkon avulla. Koska erillisiä antureita ei tarvita, on sovellus helppo ottaa käyttöön erilaisissa koneissa.

Jos sovellus havaitsee koneen liikkumisnopeuden menevän alle vähimmäistuotantonopeuden, alkaa sovellus laskea häiriöaikaa. Jos koneen nopeus pysyy alle vähimmäistuotantonopeuden yli häiriötilan alkamiskynnysajan, käynnistyy sovelluksessa ns. häiriötila. Häiriötilan tapahtuessa sovellus pyytää käyttäjää valitsemaan syyn koneen häiriöajalle eli syyn miksi käyttäjä ei ole työnteossa tai pysty työntekoon.

Projekti koostuu mobiililaitteella toimivasta sovelluksesta, jossa on käyttöliittymästä ja toiminnallisuuksista vastaava frontend ja siihen sidottuna backend-toimintoja, kuten käyttäjän autentikointi ja tiedon lähetys sekä vastaanotto. Tämä opinnäytetyö käsittelee projektin frontend-toteutusta. Frontend-puolen toteutti kahden hengen suuruinen kehitystiimi.

Opinnäytetyö selittää pääpiirtein React-ohjelmakirjaston ja miten React Native -ohjelmistokehys mahdollistaa Reactin käytön mobiilisovelluskehityksessä. Työn tarkoituksena oli toteuttaa toimiva pieni mobiilisovellus mahdolliselle asiakkaalle, todeta Reactin uusien ominaisuuksien, hookkien ja konteksti-tilanhallintarajapinnan sekä etenkin React Nativen, toimivuus mobiilisovelluskehityksessä.

2 SOVELLUKSEN SUUNNITTELU

Sovelluksen suunnitteluosiossa esitellään projektin vaatimukset ja sovelluksen toteuttamiseen valitut teknologiat.

2.1 Vaatimukset

Yleisenä vaatimuksena projektille oli käyttää moderneja ohjelmistoteknologioita ja ohjelmistotuotannon menetelmiä. Projektin kehitystiimille annettiin sovelluksen toiminnalliset ja kehityskohdevaatimukset, mutta kehitystiimi sai itse valita teknologiat, joilla sovellus toteutettiin.

2.1.1 Päätoiminto

Sovellusta käytetään mobiililaitteessa, joka sijaitsee tarkkailtavassa koneessa. Sovellus määrittelee pilvestä saaduin ehdoin sekä mobiililaitteesta saatavan liikenopeuden perusteella, onko kone tuotanto- vai häiriötilassa. Häiriötilan alkaessa eli, kun käyttäjä lopettaa aktiivisen työnteon, täytyy käyttäjän pystyä valitsemaan syy häiriöajan syntyyn. Valinnan jälkeen koneen tunniste, häiriöajan syy ja kesto sekä mobiililaitteen sijainti ja tunnistetiedot lähetetään pilveen.

Jos käyttäjä aloittaa työnteon antamatta syytä häiriöajalle, ilmoittaa sovellus itse pilveen syyksi tuntematon syy. Tuntematon syy lähetetään, kuitenkin vasta sovellusta käyttävän yrityksen määrittelemän häiriöajan syyn ilmoituksen kynnysajan jälkeen. Kynnysajan tarkoitus on antaa käyttäjälle vielä hetkellinen mahdollisuus häiriöajan syyn ilmoittamiseen, jos hän ei sitä vielä jostain syystä ennen työnteon jatkamista tehnyt. Sovelluksen päätoiminnon kierto jatkuu, kunnes sovellus suljetaan tai käyttäjä kirjautuu ulos sovelluksesta.

2.1.2 Toimiminen Androidilla ja natiivit rajapinnat

Sovelluksen pitää toimia Android-käyttöjärjestelmäisillä mobiililaitteilla ja pystyä käyttämään sen natiiveja rajapintoja. Sovelluksen pitää saada mobiililaitteen sijainti, tallettaa ja ladata tietoa muistiin sekä saada tietoonsa mobiililaitteen laitekohtainen tunnus.

2.1.3 Toimiminen taustatilassa

Olettamuksena on, että mobiililaite on koneessa jatkuvasti virtalähteeseen kytkettynä ja sovellus etualalla aktiivisena. Etualalla oloa ei kuitenkaan aina voi taata, sillä käyttäjä saattaa esimerkiksi asettaa mobiililaitteen lukitusnäyttöön tai käyttää toista sovellusta. Tämän kaltaisten tilanteiden takia sovelluksen täytyy toimia myös taustatilassa. Sovelluksen ollessa taustatilassa käyttäjän toimia vaativat toimet täytyy ilmoittaa järjestelmäilmoituksilla.

2.1.4 Tiheä ja tarkka sijainnin seuranta

Sovelluksen päätoiminto perustuu mobiililaitteen sijainnista saataviin tietoihin. Sijaintitiedoista saatavat mobiililaitteen liikkumisnopeus ja koordinaatit lähetään analysoitavaksi pilveen häiriöajan ilmoittamisen yhteydessä.

Sijaintitiedon täytyy olla tiheästi saatavaa, jotta muutamankin minuutin mittaiset häiriöajat saadaan otettua huomioon. Sijaintitiedon täytyy olla myös tarkkaa eli paikkansapitävää, joten sijainti haetaan käyttäen hakuhetkellä tarkinta mahdollista sijainnin määrittämistapaa. Nämä tavat ovat laitteen GPS, Wi-Fi ja mobiiliverkon käyttö.

Epätarkka sijaintitieto saattaa johtaa koneen liikkumisnopeuden näkymiseen todellista pienempänä tai suurempana. Kone voitaisiin virheellisesti määritellä jopa nopeasti liikkeellä olevaksi, vaikka se todellisuudessa olisi paikallaan. Tämän takia kaikki sijaintitieto, jossa tarkkuus vääristyy yli kymmenen metriä, hylätään luokittelemalla liikkumisnopeus sillä hetkellä nollaksi.

2.1.5 Sydämenlyöntitoiminto

Sovellus lähettää pilveen sitä käyttävän yrityksen määrittämän ajan välein tietoa mobiililaitteen sen hetkisestä tilasta eli sovellus tekee niin sanotusti sydämenlyöntejä. Lähetettävät tiedot ovat laitteen koordinaatit ja tieto siitä, onko kone tuotanto- vai häiriötilassa. Lähetys on ikään kuin ilmoitus onko mobiilisovellus päällä eli elossa. Lähetysten avulla voidaan tarkastella sovelluksen käyttöajan tuotanto- ja häiriötilojen suhdetta, eikä niinkään häiriöajan syitä, kuten käyttäjän itse antamissa häiriöaikailmoituksissa.

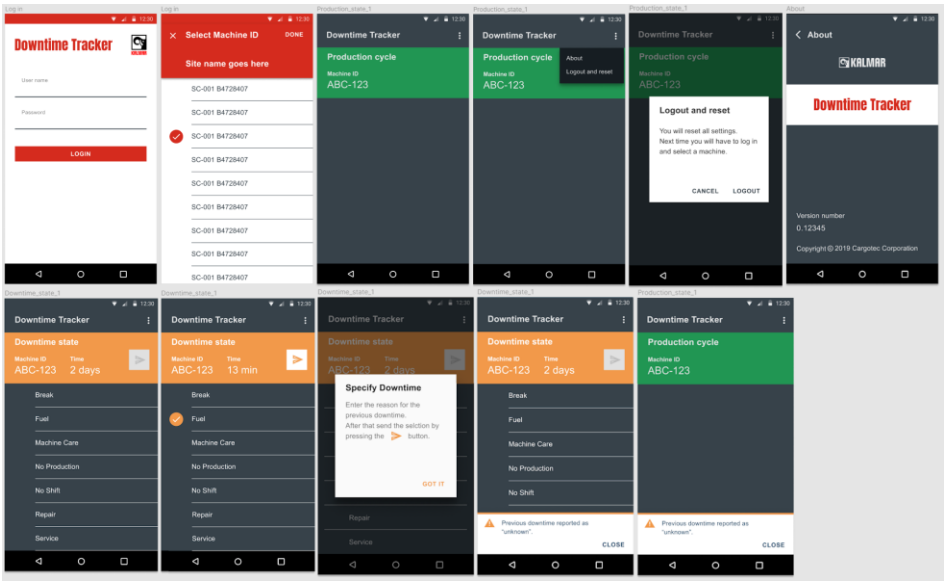
2.1.6 Käyttäjän autentikointi

Käyttäjän käyttäjä- ja päivitystunnus täytyy tallettaa onnistuneen sisäänkirjautumisen jälkeen mobiililaitteen muistiin. Ensimmäisen sisäänkirjautumisen jälkeen ja sovelluksen käynnistyessä käyttäjätunnus tarkistetaan ja tarvittaessa päivittää päivitystunnuksella pilven kautta. Koska tunnukset tallennetaan mobiililaitteen muistiin ei sisäänkirjautumista käyttäjän puolesta tarvitse suorittaa kuin vain kerran, kunhan käyttäjä ei kirjaudu välissä ulos. Käyttäjätunnus tarkistetaan ja tarvittaessa päivitetään myös kutsuja tehtäessä pilveen, jotta varmistutaan vielä sisäänkirjautumisen jälkeenkin käyttäjän autentikointi.

2.1.7 Käyttöliittymä

Kehitystiimi sai käyttöliittymäsuunnittelijalta valmiit suunnitelmat käyttöliittymän ulkoasusta (kuva 1). Käyttöliittymän haluttiin vastaavan visuaalisesti Material Design -ulkoasua ja noudattaa sen käytäntöjä. Material Design on Googlen muotokieli, jota esimerkiksi Android käyttää käyttöliittymissään.

Material Design Googlen mukaan: “Material is an adaptable system of guidelines, components, and tools that support the best practices of user interface design. Backed by open-source code, Material streamlines collaboration between designers and developers, and helps teams quickly build beautiful products.” (Material Design)



KUVA 1. Käyttöliittymän ulkoasusuunnitelma

2.2 React

React on Facebookin luoma JavaScript-ohjelmakirjasto verkkokäyttöliittymien luontiin. Reactin yksi eduista on sen komponenttipohjainen rakenne. Komponenteilla koodi voidaan paloitella loogisiin ja helposti uudelleen käytettäviin osiin, jotka tarvittaessa päivittävät itsensä muutoksien tapahtuessa sovelluksessa. (ReactJs)

2.2.1 Luokkakomponentti

React luokkakomponentit pitävät sisällään oman loogisen tilansa. Tilassa voi olla esimerkiksi tieto onko käyttäjä kirjautunut vai ei. (ReactJs. State and Lifecycle) Luokkakomponentit voivat ottaa vastaan toisten komponenttien tiloja ja tietoa eli propseja. Prop voi olla esimerkiksi teemaväri tai funktio toisesta komponentista. (ReactJs. Components and props) Alla on esimerkkikoodi luokkakomponentista (kuva 2).

```
import React, { Component } from 'react'

export default class ClassComponentExample extends Component {
    render() {
        return (
            <div>This is class component</div>
        )
    }
}
```

KUVA 2. Luokkakomponentti

Luokkakomponenteilla on myös oma elinkaarensa, joka koostuu elinkaaritoiminnoista. Tarvittaessa toimintoihin voidaan määritellä funktioita, jotka laukeavat muutoksien tapahtuessa komponentissa sen elinkaaren aikana. (ReactJs. State and Lifecycle)

Esimerkiksi luokkakomponentin elinkaaren componentDidMount-alkutoimintoon, voi määritellä aikavälilaskurin aloituksen. Komponentin elinkaaren päättyessä componentWillUnmount-lopetustoimintoon tässä tapauksessa määriteltäisiin aikavälilaskurin poistaminen muistivuodon estämiseksi (kuva 3). Esimerkin luokkakomponentin tilassa on päivämäärämuuttuja, jota laskuri päivittää sekunnin välein.

```
import React, { Component } from 'react'

export default class ClassComponentWithStateAndLifecycleExample extends Component {
    constructor(props) {
        super(props);
        this.state = { date: new Date() };
    }

    componentDidMount() {
        this.timerId = setInterval(() => {
            this.tick()
        }, 1000);
    }

    componentWillUnmount() {
        clearInterval(this.timerId);
    }

    tick() {
        this.setState({ date: new Date() });
    }

    render() {
        return (
            <div>Time now: {this.state.date.toLocaleTimeString()}</div>
        )
    }
}
```

KUVA 3. Aikavälilaskuri luokkakomponentilla toteutettuna

2.2.2 Funktionaalinen komponentti

Reactissa on luokkakomponenttien lisäksi funktionaalisia komponentteja (kuva 4). Niitä sanotaan myös tilattomiksi funktionaalisiksi komponenteiksi. Ne ovat toiminnoiltaan yksinkertaisempia ja ne eivät pitemmän nimensä mukaan omaa loogista tilaa. Niillä ei myös ole luokkakomponentin kaltaista elinkaarta ja näin ollen niiden kanssa ei voi käyttää elinkaaritoimintoja. (Robin Wieruch 2019)

```
1  import React from 'react'
2
3  export default function FunctionalComponentExample() {
4      return (
5          <div>This is functional component</div>
6      )
7  }
```

KUVA 4. Funktionaalinen komponentti

2.2.3 React hookit

Ennen React 16.8 -päivitystä elinkaaritoimintoja ja tiloja tarvitsevat toimet täytyi toteuttaa luokkakomponenteilla. Usein selkeämmäksi miellettyjä funktionaalisia komponentteja pystyi käyttämään vain yksinkertaisissa toimissa.

Vuonna 2019 helmikuun 6. päivä julkaistu React 16.8 -päivitys toi kuitenkin mukanaan React hookit. Hookit mahdollistavat tilan ja elinkaaritoimintojen kaltaisten ominaisuuksien käytön funktionaalisissa komponenteissa. (ReactJs. React v16.8: The One With Hooks; Introducing Hooks)

Hookit toimivat vain funktionaalisissa komponenteissa. Luokkakomponentteja ja funktionaalisia komponentteja hookkien kera voi silti edelleenkin käyttää keskenään ongelmitta. Tämän opinnäytetyön projektin React-koodi on kuitenkin toteutettu käyttäen vain funktionaalisia komponentteja hookkien kera oppimis- ja kokeilumielessä.

Hookkien etu ilmenee verrattaessa esimerkiksi aikavälilaskurin toteutusta hookkien avulla luokkakomponentti toteutukseen (kuva 5).

```jsx
import React, { useState, useEffect } from 'react'

export default function ClockFunctionalComponent() {
    const [date, setDate] = useState(new Date());

    useEffect(() => {
        const timerId = setInterval(() => {
            tick();
        }, 1000);
        return () => clearInterval(timerId)
    }, [])

    function tick() {
        setDate(new Date());
    }

    return (
        <div>
            Time now: {date.toLocaleTimeString()}
        </div>
    )
}
```

```jsx
import React, { Component } from 'react'

export default class ClockClassComponent extends Component {
    constructor(props) {
        super(props);
        this.state = { date: new Date() };
    }

    componentDidMount() {
        this.timerId = setInterval(() => {
            this.tick()
        }, 1000);
    }

    componentWillUnmount() {
        clearInterval(this.timerId);
    }

    tick() {
        this.setState({ date: new Date() });
    }

    render() {
        return (
            <div>
                Time now: {this.state.date.toLocaleTimeString()
            </div>
        )
    }
}
```

KUVA 5. Funktionaalisen ja luokkakomponentin vertailu

Kuten esimerkistä selviää, hookit lyhentävät ja selkeyttävät koodia. Huomataan myös, että hookit mahdollistavat useiden elinkaaritoimintojen liittämisen yhteen hookkiin nimeltä useEffect. Se vastaa luokkakomponenteissa käytettyjä elinkaaritoimintoja componentDidMount, shouldComponentUpdate ja componentWillUnmount. Elinkaaritoimintojen liittäminen yhteen selkeyttää koodia entisestään. (ReactJs. Using the Effect Hook)

React hookkejen oikeaoppisen käytön tarkistaminenkin on helppoa, sillä React on tehnyt ESLint-kytkennän nimeltä eslint-plugin-react-hooks. Kytkennän voi kytkeä ohjelmointiprojektiinsa ja se automaattisesti tarkistaa hookkeihin liittyviä ohjelmointivirheitä. (ReactJS. Rules of Hooks)

2.2.4 Kustomoidut hookit

React mahdollistaa omien eli kustomoitujen hookkien luonnin. Reactin sivuilla on ohjeita hyvien käytänteiden noudattamiseen hookkien luonnissa ja Reactin hookki ESLint-kytkentä tarkistaa myös kustomoidut hookit. Kustomoitujen hookkien periaatteena on niiden helppo jaettavuus ja uudelleenkäytettävyys.

Alla olevassa esimerkissä aikavälilaskurin logiikka on luotu omaan kustomoituun hookkiinsa (kuva 6). Hookki suorittaa aikavälilaskurilogiikan ja palauttaa päivämäärän.

```
1   import { useState, useEffect } from 'react'
2
3   export default function useDate() {
4       const [date, setDate] = useState(new Date());
5
6       useEffect(() => {
7           const timerId = setInterval(() => {
8               tick();
9           }, 1000);
10          return () => clearInterval(timerId)
11      }, [])
12
13      function tick() {
14          setDate(new Date());
15      }
16
17      return date
18  }
```

KUVA 6. Aikavälilaskuri-kustomoitu hookki

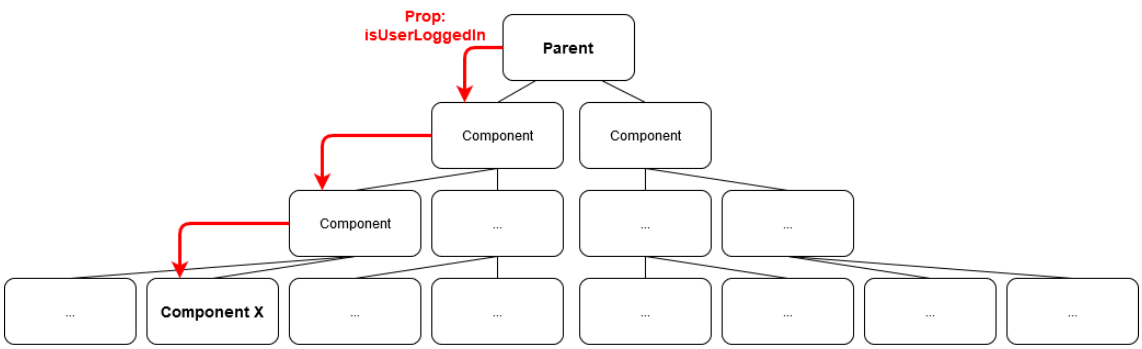
Aikavälilaskuri-kustomoidun hookkia voi nyt helposti käyttää missä tahansa funktionaalisessa komponentissa (kuva 7). Jälleen huomataan, että funktionaalisten komponenttien ja hookkien avulla koodista saa selkeää.

```
1   import React from 'react'
2   import useDate from '../customHooks/useDate'
3
4   export default function FunctionalComponentWithHooksExample2() {
5       const date = useDate();
6
7       return (
8           <div>Time now: {date.toLocaleTimeString()}</div>
9       )
10  }
```

KUVA 7. Aikavälilaskuri käyttäen kustomoitua hookkia

2.2.5 Tilanhallinta Reactissa

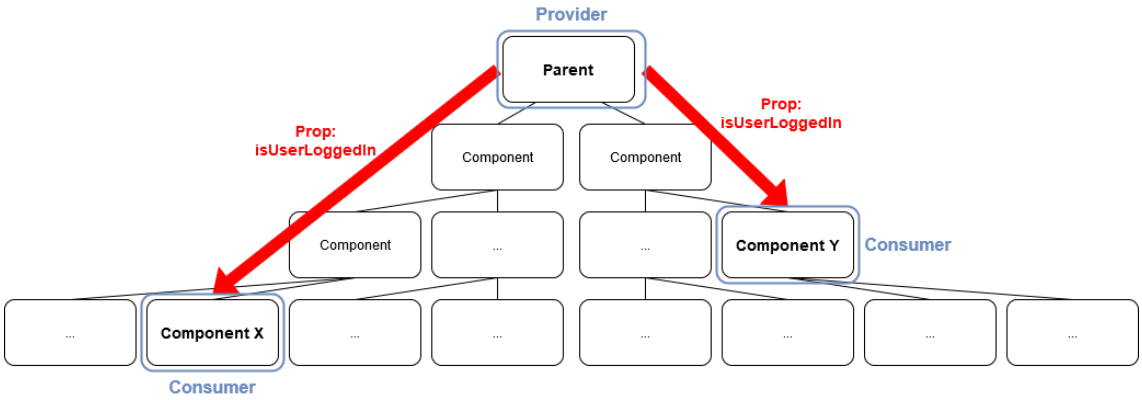
Reactissa komponenteilla voi olla omia loogisia tiloja ja komponentit voivat ottaa vastaan toisten komponenttien tiloja ja tietoa eli propseja. Reactissa toista komponenttia kutsuvaa komponenttia sanotaan kutsuttavan komponentin vanhemmaksi ja kutsuttua komponenttia vanhemman lapseksi. Lapsikomponentti voi myös olla vanhempi toiselle komponentille. Komponenttien ketjun voi mieltää komponenttipuuksi, jossa halutut propsit tarjotaan ylhäältä alaspäin eli vanhemmalta lapselle aina haluttuun komponenttiin saakka (kuva 8).



KUVA 8. Propsien tarjoaminen ilman tilanhallintatyökalua

React-projektin kasvaessa tarjottujen propsien ja komponenttien määrä komponenttipuussa kasvaa ja koodi saattaa muuttua helposti vaikeasti luettavaksi. Tätä varten kolmannen osapuolen tiimit ovat luoneet itse tilanhallintatyökaluja, kuten MobX ja Redux.

Tilanhallintatyökalujen tarkoituksena on poistaa tarve propsien tarjoamiseen komponentilta toisten komponenttien läpi. Tilanhallintatyökalujen avulla useasti käytettävää tietoa voidaan jakaa suoraan komponenttiin. Projekteissa tämän kaltaista useasti käytettävää tietoa voi olla esimerkiksi tieto siitä, onko käyttäjä kirjautunut vai ei (kuva 9).



KUVA 9. Propsien tarjoaminen tilanhallintatyökalulla

2.2.6 React konteksti -tilanhallintarajapinta

React 16.3.0 -päivitys toi Reactiin oman tilanhallintatavan nimeltä konteksti. Konteksti poistaa tarpeen kolmannen osapuolen valmistaman tilanhallintatyökalu. Päivitys julkaistiin maaliskuun 29. päivä 2018 eli vasta lähes kolmen vuoden päästä Reactin julkaisusta. (ReactJs. React v16.3.0: New lifecycles and context API)

Kontekstin käytöönotossa luodaan ensin itse konteksti. Kontekstiin voidaan tehdä mutkikkaitakin logiikoita käyttäen esimerkiksi elinkaaritoimintoja ja se voi tarjota niin muuttujia, kuin funktioitakin eteenpäin.

Esimerkin kontekstissa (kuva 10) määritellään yksinkertaisuuden vuoksi vain kaksi värin sisältävää muuttujaa, jotka annetaan tarjoajaan eli provideriin.

```
1   import React, { createContext } from 'react'
2
3   export const ColorTheme = createContext()
4
5   export default function ColorThemeProvider(props) {
6       const primaryColor = 'red'
7       const secondaryColor = 'blue'
8       return (
9           <ColorTheme.Provider value={{ primaryColor, secondaryColor }}>
10              {props.children}
11          </ColorTheme.Provider>
12      )
13  }
```

KUVA 10. Kontekstin luonti

Komponentit, joille halutaan pääsy kontekstin muuttujiin, kutsutaan kuluttajiksi ja ne ympäröidään tarjoajakomponentilla. Esimerkin tapauksessa ColorThemeProvider toimii tarjoajakomponenttina (kuva 11).

```
import React from 'react';
import {
  ClassComponentExample,
  FunctionalComponentExample,
  ClassComponentWithStateAndLifecycleExample,
  FunctionalComponentWithHooksExample,
  FunctionalComponentWithHooksExample2,
  FunctionalComponentExampleWithContext
} from './components'
import ColorThemeProvider from './contexts/ColorTheme'

export default function App() {
  return (
    <div className="App">
      <ColorThemeProvider>
        <ClassComponentExample />
        <FunctionalComponentExample />
        <ClassComponentWithStateAndLifecycleExample />
        <FunctionalComponentWithHooksExample />
        <FunctionalComponentWithHooksExample2 />
        <FunctionalComponentExampleWithContext />
      </ColorThemeProvider>
    </div>
  );
}
```

KUVA 11. Kontekstin tarjoaminen komponenteille

Kuluttajakomponentit pääsevät nyt käyttämään kontekstin muuttujia tarjoajakomponentin kautta. Jos kuluttajakomponenttina on funktionaalinen komponentti, voidaan arvot saada helposti käyttäen useContext-hookkia (kuva 12).

```
import React, { useContext } from 'react'
import { ColorTheme } from '../contexts/ColorTheme'

export default function FunctionalComponentExampleWithContext() {
    const { primaryColor } = useContext(ColorTheme)
    return (
        <div style={{ color: primaryColor }}>
            This is functional component
        </div>
    )
}
```

KUVA 12. Kontekstin käyttäminen

Yllä olevassa esimerkissä luodusta kontekstista otetaan muuttuja primaryColor, jonka avulla määritellään komponentin tekstin väri. Tällä tavoin saataisiin esimerkiksi koko projektin komponentteihin kätevästi sama väriteema. Tarvittaessa väriteeman värien vaihtaminen onnistuisi helposti muuttamalla vain kontekstin muuttujia.

2.3 React Native

React on pääsääntöisesti verkkokäyttöliittymien tekemiseen luotu ohjelmistokirjasto. Facebook mahdollistaa myös natiivin Android- ja iOS-mobiilikehityksen React Native -ohjelmistokehyksellä.

React Native on Reactia, mutta siinä JavaScript prosessoidaan mobiililaitteen taustaprosessissa mobiililaitteen käyttöjärjestelmälle ymmärrettävään muotoon. Näin päästään käyttämään mobiililaitteen käyttöjärjestelmän natiiveja rajapintoja, kuten kameraa ja muistinhallintaa. Lisäksi päästään käyttämään natiiveja käyttöliittymäkomponentteja, kuten näppäimistöä ja nappeja. (React Native)

React Nativen ansiosta voi JavaScript-verkkosivuohjelmoinnin osaava ruveta kehittämään sovellusta mobiililaitteelle opettelematta uutta ohjelmointikieltä.

```
1   import React from 'react';
2   import { View } from 'react-native';
3
4   export default function ReactNativeExample() {
5       <View>This is component for React Native</View>
6   }
```

KUVA 13. React Native esimerkki

React Nativen yksi pääeduista on sen alustariippumaton sovelluskehitys mahdollisuus eli sama JavaScript-koodi toimii sekä Android- ja iOS-käyttöjärjestelmille. Perinteisesti natiivissa mobiiliohjelmointikehityksessä molemmille käyttöjärjestelmille täytyi toteuttaa omat sovelluksensa, Androidille Java-ohjelmointikielellä ja iOS:lle Swift-ohjelmointikielellä.

Tämän opinnäytetyön projektin ohjelmistokehykseksi valittiin React Native, sillä se mahdollisti mobiililaiteen käyttöjärjestelmän natiivien rajapintojen käytön. Lisäksi molemmilla kehitystiimin jäsenillä oli aiempaa kokemusta Reactin käytöstä. Java-ohjelmointikielen kokemusta oli vähemmän ja, koska vaatimuksena oli Android-alustaisilla mobiililaitteilla toimiva sovellus, ei Swiftiä voinut käyttää.

2.4 Googlen sijaintirajapinta Androidille

Google antaa monia eri rajapintoja ohjelmistokehittäjien käyttöön. Googlella on myös rajapintakokoelma Androidille nimeltä APIs for Android. Kokoelma sisältää sijainnin seurannan mahdollistavan rajapinnan, jonka toiminta perustuu GPS:ään, Wi-Fi:iin ja mobiiliverkkoon. (Google)

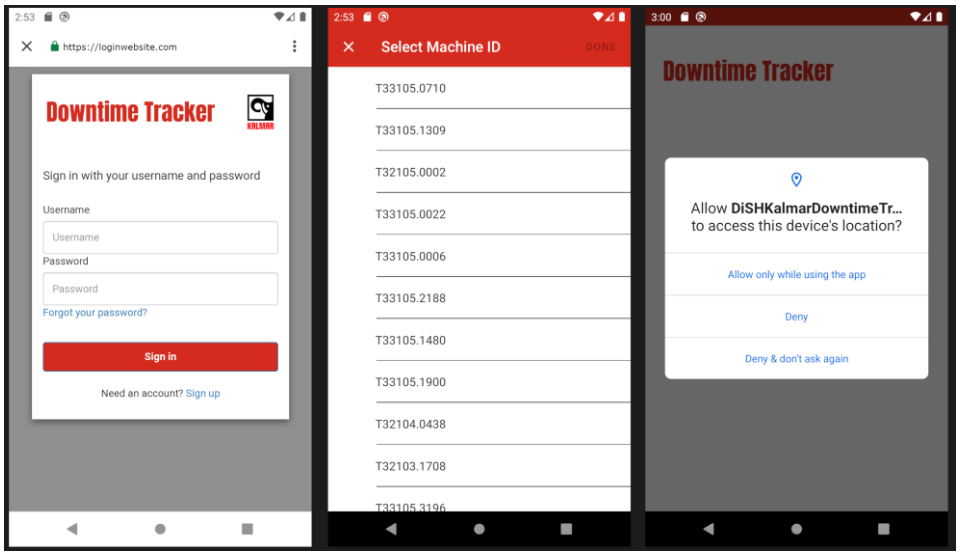
Sijaintirajapinnan sijaintitiedoista pystyy saamaan mobiililaiteen liikkumisnopeuden, koordinaatit, aikaleiman ja sijaintitiedon tarkkuuden metreissä.

3 SOVELLUKSEN TOTEUTUS

Sovelluksen toteutusosiossa esitellään sovelluksen tärkeiden toimintojen toteutus.

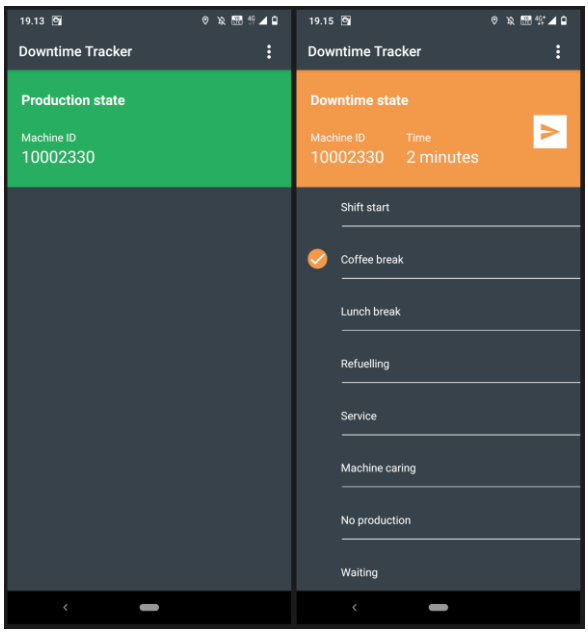
3.1 Toiminta

Tämän opinnäytetyön sovelluksen voi jakaa autentikointi- ja päätoiminto-osuuteen. Autentikointiosuus sisältää sisäänkirjautumisen, koneen valinnan ja poiston sekä sijainnin antoluvan tarkistamisen ja tarvittaessa pyytämisen (kuva 14).



KUVA 14. Autentikointiosuuden näkymät

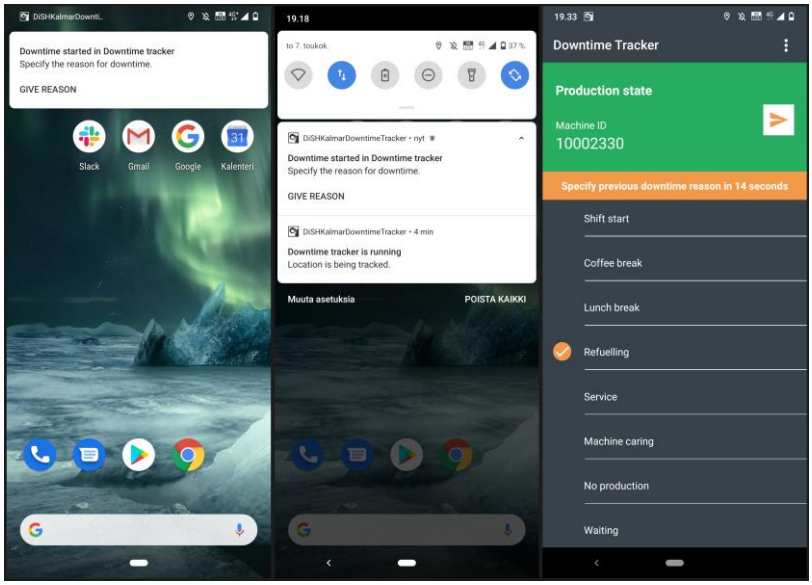
Autentikoinnin jälkeen seuraa päätoiminto-osuus. Päätoiminto-osuuden alussa pilvestä haetaan sovellusta käyttävän yrityksen määrittelemät asetukset. Sen jälkeen käynnistetään sijainnin seuranta ja aikavälilaskurilla toimiva sydämenlyöntitoiminto. Päätoiminnon alussa sovellus on tuotantotilanäkymässä (kuva 15).



KUVA 15. Päätoiminto-osuus: tuotanto- ja häiriötilanäkymä

Mobiililaitteen sijaintiedon tarkkuuden ja liikkumisnopeuden perusteella määritellään jatkuvasti, onko kone tuotanto- vai häiriötilassa. Jos kone menee häiriötilaan, aloitetaan aikavälilaskuri, joka pilvestä haetun häiriötilan aloittamisen kynnysajan päästä siirtää sovelluksen häiriötilanäkymään, ellei koneen liikkumisnopeus ylitä vähimmäistuotantonopeutta. Aikavälilaskuri jatkaa häiriöajan pituuden laskemista. Tässä käyttäjän pitäisi valita syy häiriöajalle listasta, jonka on määritellyt sovellusta käyttävä yritys pilveen. Häiriöajan syyn valinnan jälkeen aikavälilaskuri nollataan sekä pysäytetään ja sovellus siirtyy tuotantotilanäkymään. Tämä päätoiminto-osuuden toiminnallisuus jatkuu sovelluksen käytön ajan.

Päätoiminto-osuudessa on kaksi erikoistilannetta. Ensimmäinen tapahtuu, jos sovellus on taustalla sovelluksen siirtyessä häiriötilaan. Tällöin sovellus antaa järjestelmäilmoituksen, jossa käyttäjää pyydetään antamaan syy häiriöajalle sovelluksen kautta. Järjestelmäilmoituksen tarkoitus on saada käyttäjän huomio. Järjestelmäilmoitus ilmestyy näkyviin muutamaksi sekunniksi isona mobiililaitteen näytön yläosaan ja siirtyy sitten pieneksi mobiililaitteen järjestelmäpalkkiin. Toinen erikoistapaus syntyy käyttäjän virheestä, kun käyttäjä ei anna syytä häiriöajalle ja siirtyy takaisin tuotantotilaan toimillaan eli käytännössä jatkamaan työntekoa. Tällöin sovellus pyytää käyttäjää antamaan syyn häiriöajalle jälkikäteen sovellusta käyttävän yrityksen määrittämässä aikavälissä. Jos syytä ei anneta, lähetetään häiriöajan syynä tuntematon syy.



KUVA 16. Päätoiminto-osuus: erikoistilanteet

3.2 Käyttöliittymän vaatimusten täyttö

Jotta sovellus täytti Googlen Material Designin ehdot, otettiin käyttöön suosittu käyttöliittymäkirjasto React Native Paper. Kyseinen käyttöliittymäkirjasto sisältää valmiit Material Designia noudattavat ja React Nativella toimivat käyttöliittymäkomponentit, joihin oli helppo soveltaa käyttöliittymäsuunnitelmien väriteema. (GitHub. React Native Paper)

3.3 Näkymien välinen liikkuminen

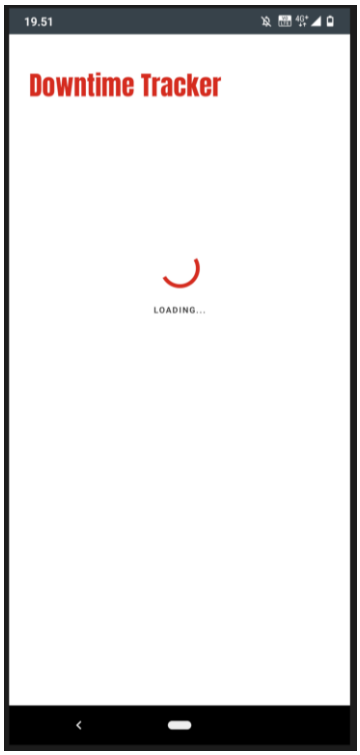
Mobiilisovelluksissa sujuvalla näkymien välisellä liikkumisella on suuri merkitys käyttäjäkokemukseen. Vaikka opinnäytetyön sovelluksessa on vain viisi eri näkymää, täytyi näkymien välistä liikkumista miettiä tarkoin. Toteutusta vaikeutti se, että kirjautuminen tapahtuu sovelluksen kautta aukeavalla ulkoisella verkkosivulla. Oli myös otettava huomioon, että vain kirjautunut käyttäjä päästetään tiettyihin näkymiin.

Sovelluksen käynnistyttyä aukeaa latausnäkymä, jossa laitteen muistista haetaan käyttäjätunnus, valitun koneen tunnus ja tarkistetaan, onko käyttäjä antanut

sovellukselle sijainnin käyttöluvan. Saatujen tietojen löytymisen tai niiden puuttumisen perusteella sovellus avaa käyttäjälle oikean näkymän.

Jos muistista ei löydy käyttäjätunnusta, avautuu sisäänkirjautumisverkkosivu. Sisäänkirjautumisverkkosivu palauttaa onnistuneen kirjautumisen jälkeen sovellukselle muuttujan arvolla tosi. Tällöin voidaan käyttäjä ohjata suoraan seuraavaan näkymään, ilman että sovelluksen tarvitsee tehdä muita sivutoimia tai turhaan enää näyttää käyttäjälle latautumisnäkymää. Tämän kaltainen toteutus virtaviivaistaa käyttäjäkokemusta.

Jos käyttäjä on jo aiemmin kirjautunut sisään sovellukseen, varmistetaan käyttäjätunnuksen avulla käyttäjän autentikointi. Kaikki nämä edellä mainitut sovelluksen alkuvaiheen sivutoimet tapahtuvat hetken käyttäjälle näkyvän latausnäkymän aikana (kuva 17).



KUVA 17. Latausnäkymä

3.4 Sijaintiluvan tarkistus ja pyytäminen

Android-käyttöjärjestelmä ei salli joidenkin toimintojen käyttöä mobiililaitteilla ilman, että käyttäjä on antanut niille erikseen luvan. Näitä erikseen pyydettäviä lupia ovat Androidin määrittämät ns. vaaralliset luvat. Tämä vaatii erillistä huomiota sovelluksen toteutuksessa ja kehittäjän tarvitsee itse miettiä milloin lupien kysyntä pitäisi suorittaa. Vaaralliset luvat kuuluvat toimintojen piiriin, jotka pääsevät käsiksi käyttäjän henkilökohtaisiin tietoihin. Vaarallisten lupien toimintoja ovat esimerkiksi kameran käyttö, kalenteri ja sijainnin käyttö. (Android. Permissions overview; Stackon Technologies 2017)

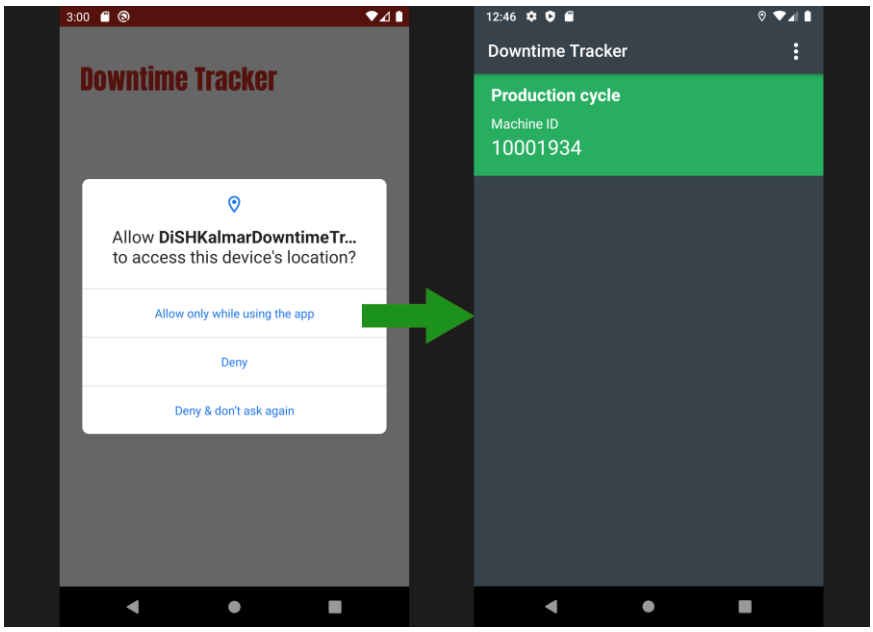
Opinnäytetyön projektin kohdalla sovellus vaatii lupaa käyttäjältä tarkan sijainnin käyttöön eli ACCESS_FINE_LOCATION-lupaa. Kyseinen lupa antaa nimensä mukaisesti pääsyn laitteen tarkkaan sijaintiin. (Android. Manifest.permission)

React Native sisältää valmiin rajapinnan lupien pyytämiseen Androidilla. Alla olevassa esimerkissä on sovelluksessa käytettävä sijaintiluvan pyyntö -funktio (kuva 8)

```
42  const requestLocationPermission = useCallback(async () => {
43    try {
44      const result = await PermissionsAndroid.request(PermissionsAndroid.PERMISSIONS.ACCESS_FINE_LOCATION, undefined);
45      if (result === PermissionsAndroid.RESULTS.GRANTED) {
46        log('Location permission is given, proceeding');
47        setLocationPermissionGiven(true);
48      }
49      if (result === PermissionsAndroid.RESULTS.NEVER_ASK_AGAIN) {
50        log('Location permission denied and futher requests blocked');
51        setLocationPermissionGiven(false);
52      }
53      if (result === PermissionsAndroid.RESULTS.DENIED) {
54        log('Location permission denied, it must be allowed to proceed');
55        setLocationPermissionGiven(false);
56      }
57      return result;
58    } catch (error) {
59      log(error.message);
60      setLocationPermissionGiven(false);
61      return false;
62    }
63  }, []);
```

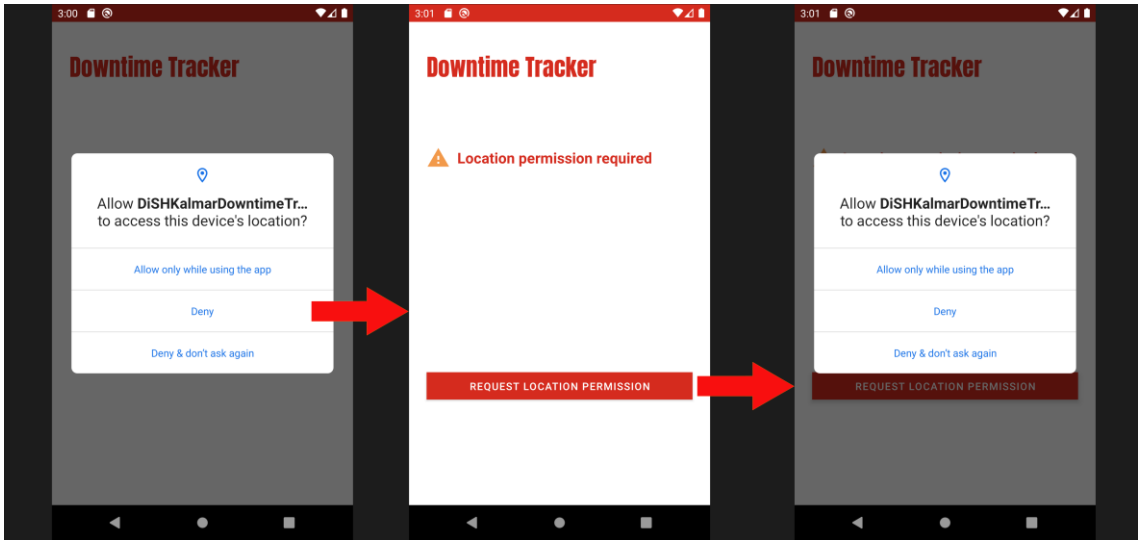
KUVA 18. Sijaintiluvan pyyntö React Nativessa

Funktiota käytetään, jos sijaintilupaa ei ole vielä annettu sovelluksen yrittäessä mennä toiminto-osuuden päänäkymään. Funktio aukaisee Androidin oman natiivin pyyntödialogin ja käsittelee käyttäjän valinnan (kuva 9). Alla olevasta kuvasta ilmenee sovelluksen toiminta käyttäjän hyväksyessä sijaintiluvan pyynnön.



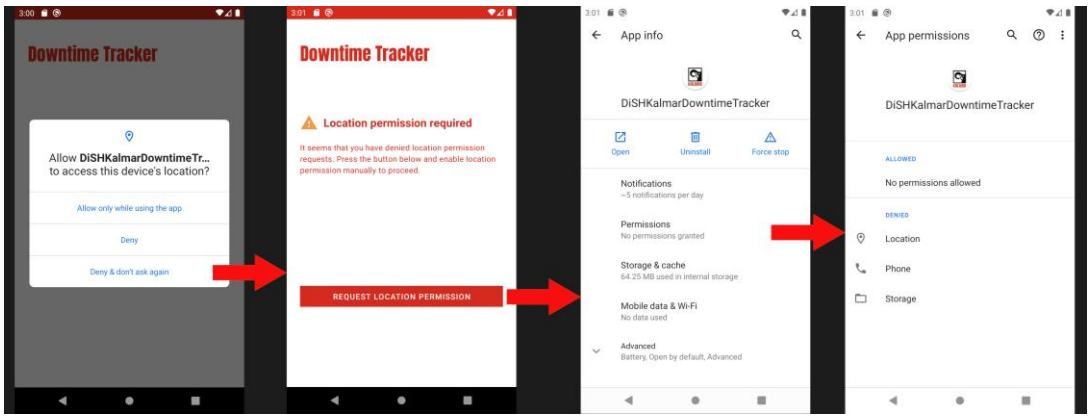
KUVA 19. Sijaintiluvan salliminen

Kun käyttäjän hyväksyy sijaintiluvan täyttyvät sovelluksen ehdot ja sovellus siirtyy päänäkymään. Jos käyttäjä ei hyväksy sijaintilupaa toimitaan alla olevan kuvan mukaisesti (kuva 20).



KUVA 20. Sijaintiluvan kieltäminen

Sovellus ei päästä käyttäjää jatkamaan yllä olevassa kuvassa näkyvistä näkymistä eteenpäin ilman sijaintiluvan hyväksymistä. Jos käyttäjä valitsee, ettei sovellus saa enää pyytää sijaintilupaa, toimitaan alla olevan kuvan mukaisesti (kuva 21).



KUVA 21. Kielletyn sijaintiluvan pyytämisen kiertäminen

Tässä tapauksessa käyttäjä ohjeistetaan manuaalisesti sallimaan sijaintilupa. Nyt sijaintiluvan pyyntö -nappi avaa suoraan sovelluksen asetukset, sillä Android estää pyyntödialogin avaamisen. Koska sijainnin saanti on välttämätön osa opinnäytetyön sovelluksen toimintaa, on käyttäjä ikään kuin pakotettava sijaintiluvan antamiseen sovelluksen käyttämiseksi.

3.5 Muistin käyttö

Android sovelluksissa mobiililaitteen muistin käyttöön ei tarvitse pyytää käyttäjältä lupaa, sillä sovelluksella ei ole pääsy henkilökohtaisiin tietoihin. React Nativen tapa mobiililaitteen muistinhallintaan sovelluksen kautta on AsyncStorage -niminen rajapinta. React Nativen omilla sivuilla kuitenkin kehotetaan käyttämään React Native Communityn Async Storagea. (React Native. AsyncStorage)

React Native Community eli RNC on organisaatio, joka yllä pitää React Nativen sisältämiä paketteja ja kirjastoja. RNC koostuu React Native yhteisön jäsenistä, jotka tekevät yhteistyötä Facebookin React Native -tiimin kanssa. (GitHub. The React Native Community)
Opinnäytetyön sovelluksessa mobiililaitteen muistiin tallennettiin käyttäjätunnus ja sen päivittämiseen käytettävä päivitystunnus sekä käyttäjän valitseman koneen tunnus. Alla olevissa kuvissa on toteutus koneen tunnuksen tallettamisesta (kuva 22), lukemisesta (kuva 23) ja poistamisesta mobiililaitteen muistista (kuva 24).

```
32  const selectMachine = async (id) => {
33    try {
34      await AsyncStorage.setItem('machineId', id.toString(), () => {
35        setMachineId(id);
36        log('selectMachine success');
37      });
38    } catch (error) {
39      log('selectMachine failed', error);
40      setMachineId(null);
41    }
42  };
```

KUVA 22. Tallettaminen laitteen muistiin

```
11  useEffect(() => {
12    (async function getMachineFromAsyncStorage() {
13      setMachineContextLoading(true);
14      try {
15        await AsyncStorage.getItem('machineId').then((id) => {
16          if (id !== null) {
17            setMachineId(id);
18            log('INIT: machine found from storage');
19          } else {
20            setMachineId(null);
21            log('INIT: machine not found from storage');
22          }
23        });
24      } catch (error) {
25        log('INIT: getUserFromAsyncStorage failed', error.message);
26      }
27      setMachineContextLoading(false);
28    }());
29  }, []);
```

KUVA 23. Tallennuksen luku laitteen muistista

```
44  const deSelectMachine = async () => {
45    try {
46      await AsyncStorage.removeItem('machineId', () => {
47        setMachineId(null);
48        log('deSelectMachine success');
49      });
50    } catch (error) {
51      log('deSelectMachine failed', error);
52    }
53  };
```

KUVA 24. Talletuksen poisto laitteen muistista

3.6 Sovelluksen laajuiset muuttujat ja niiden hallinta

Sovelluksessa tilanhallintaan käytettiin Reactin kontekstirajapintaa. Alla oleva taulukko sisältää sovellukseen luodut kontekstit toimintoineen (taulukko 1).

TAULUKKO 1. Kontekstit

Nimi	Sisältö
Auth	• Käyttäjän tunnusten hallinta • Sisään- ja uloskirjautumisfunktiot
Configuration	• Sovelluksen asetusten ja häiriötilasyylistan haku pilvestä
LocationPermission	• Sovelluksen sijainninkäyttö luvan hallinta • Sijainninkäyttöluvan pyyntö -funktio
Machine	• Valitun koneen tunnuksen hallinta
MachineState	• Sydämenlyöntitoiminnon ja sijainnin seurannan aloitus • Mobiililaitteen liikkumisnopeuden seuranta • Tuotanto- ja häiriötilan määrityslogiikka

Usean kontekstin käytön syynä on sovelluksen laajuisten toimintojen ja muuttujien hallinnan selkeyttäminen. Kontekstien sovelluksen laajuisten muuttujien ja funktioiden avulla saadaan esimerkiksi sydämenlyöntitoiminto päälle ja pois päältä riippumatta siitä, missä sovelluksen näkymässä käyttäjä on. Myös sijainninseuranta ja muut toiminnot pystytään lopettamaan sovelluksen näkymästä ja tilasta riippumatta, jos esimerkiksi sovelluksen sijaintilupa evätään mobiililaitteen asetuksista.

3.7 Toimiminen taustatilassa

Suuri osa sovelluksen logiikasta perustuu aikavälilaskurien käyttöön. React Nativen sisältämä aikavälilaskuri toimii vain sovelluksen ollessa etualalla, joten projektiin ladattiin kolmannen osapuolen ohjelmistokirjasto. Kyseinen kirjasto toimii natiivimoduulin avulla, kuten moni muukin React Nativen omista ja ulkopuolisista kirjastoista. Natiivimoduuli on React Nativen tapa tuoda natiiveja toimintoja React Native -puolen käyttöön. (React Native. Native Modules) Myös ilmoitusten luontiin täytyi ladata React Nativen ulkopuolinen kirjasto.

3.8 Sijainnin seuranta taustatilassa

Projektin teko hetkellä React Native ei sisältänyt tapaa suorittaa taustatoimintoja pelkällä React Nativen sisältämillä rajapinnoilla.

3.8.1 Vaihtoehdot

Taustatoimintojen suorittamiseksi React Native -projektissa tarvitaan kohdekäyttöjärjestelmän natiivilla koodilla tehtyä natiivimoduulia.

Verkossa on monia kolmannen osapuolen ohjelmistokirjastoja, jotka sisältävät natiivintoteutuksen jollekin taustatoiminnolle. Kolmannen osapuolen ohjelmistokirjastoja saatetaan kuitenkin päivittää hitaasti, sillä ne ovat usein vain yhden hengen ylläpitämiä. Joskus taas ohjelmistokirjastot ovat suuren ohjelmointitiimin toteutuksia, mutta ne saattavat olla maksullisia.

Tämä tilanne oli React Nativelle tehtyjen taustatilassa toimivien sijainnin seurantaohjelmistokirjastojen kohdalla, joten kehitystiimi päätti olla käyttämättä kolmannen osapuolen toteutusta. Taustatilassa toimiva sijainnin seuranta päätettiin toteuttaa itse Java-ohjelmointikielellä eli Androidin natiivilla koodilla. Omassa Java-toteutuksesta luotiin natiivimoduuli, jota pystyttiin käyttämään React Native -projektin JavaScript-puolella.

3.8.2 Oma Java-toteutus

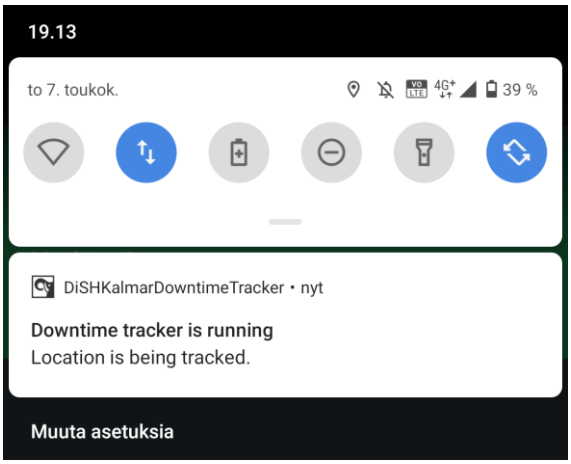
Ehkä yksinkertaisin tapa suorittaa taustatoimintoja Androidilla on käyttää siihen suunniteltua Androidin omaa JobScheduler-rajapintaa. JobScheduleria pystyy kuitenkin ajamaan vain 15 minuutin minimiaikavälillä, mikä melkein reaaliaikaisen sijainnin seurannan vaativalle sovellukselle olisi aivan liikaa.

Jotta taustalla toimiva sijainnin seuranta saatiin toteutettua ilman pitkää aikaväliä, käytettiin siihen Androidin Alarm Manager -rajapintaa, Foreground Service - ja

Background Service -toimintoja sekä Googlen APIs for Android -sijaintirajapinnan FusedProviderClient-pakettia.

Alarm Manager -rajapinta mahdollistaa taustatoiminnon aloittamisen, kunhan sovellus ei ole taustatilassa. Foreground Services -toiminnolla pystytään ajamaan toimintoja, vaikka sovellus olisi taustatilassa. Googlen tekemä FusedProviderClient-paketti sisältää toiminnon nimeltä requestLocationUpdates jatkuvalle sijainnin saannille.

Projektin sovelluksen tapauksessa Alarm Managerilla aloitettiin taustalla pyörivä Foreground Service -toiminto, joka pitää yllä requestLocationUpdates-toimintoa. Android antaa Foreground Service -toiminnon pyörimisen sovelluksen ollessa taustalla, sillä käyttäjälle näkyy siitä pysyvä ilmoitus mobiililaitteen järjestelmäpalkissa (kuva 25).



KUVA 25. Järjestelmäilmoitus taustatoiminnosta

RequestLocationUpdates-toiminnolle saa asetettua sijainninhakuasetuksia tarpeen mukaan. Hakuasetuksiksi määriteltiin tarkka sijainnin käyttö ja sekunnin pyyntöväli eli sijaintia seurattiin sekunnin tarkkuudella. Tarkalla sijainnin käytöllä laitteen sijainti haetaan hakuhetkellä tarkimpana olevasta lähteestä. Sijainnin hakulähteet ovat mobiililaitteen GPS, Wi-Fi tai mobiiliverkko. Tarkan sijainnin käytön haittapuoli on hieman suurempi akun kulutus epätarkempiin hakutapoihin verrattuna. Koska sovellus on suunniteltu suurimmaksi osaksi laturissa olevassa mobiililaitteessa käytettäväksi ja sijainnin saanti oli tärkeä osa sovelluksen toimintaa ei hieman suurempaa akunkäyttöä katsottu haitaksi.

Java-puolen toteutuksesta luotiin siis natiivimoduuli, joka mahdollisti natiivin koodin käytön React Nativen JavaScript-puolella. Natiivimoduuli antoi funktiot sijainnin seurannan aloittamiseen ja lopettamiseen sekä requestLocationUpdates-toiminnolla saadut sijaintitiedot reaaliajassa.

Natiivimoduulin avulla luotiin React Native -puolelle sijainnin seuranta -hookki, joka isRunning-nimisen muuttujan totuusarvon perusteella käynnistää tai lopettaa sijainninseurannan (kuva 26).

```
import {useState, useEffect} from 'react';
import { NativeModules, DeviceEventEmitter } from 'react-native';

// Hook to return position, works on foreground and background
export default function useNativeWatchPosition(isRunning = null) {
  const [position, setPosition] = useState({
   latitude: null, longitude: null, speed: null, timestamp: null, accuracy: null
  });

  useEffect(() => {
    if (isRunning !== null) {
      if (isRunning === true) {
        NativeModules.LocationManager.startBackgroundLocation();
      } else {
        NativeModules.LocationManager.stopBackgroundLocation();
      }
    }
  }, [isRunning]);

  useEffect(() => {
    let subscription = null;
    if (isRunning === true) {
      subscription = DeviceEventEmitter.addListener(
        NativeModules.LocationManager.JS_LOCATION_EVENT_NAME, (pos) => setPosition(pos),
      );
    }
    return () => {
      if (subscription !== null && isRunning !== null) {
        NativeModules.LocationManager.stopBackgroundLocation();
        subscription.remove();
      }
    };
  }, [isRunning]);

  return {position};
}
```

KUVA 26. Sijainnin seuranta -hookki

Sijainnin seuranta -hookin isRunning-arvo asettuu todeksi ja sijaintia ruvetaan seuraamaan, kunhan käyttäjä on kirjautunut sisään, valinnut koneen ja antanut sijainnin seuranta luvan (kuva 27).

```
useEffect(() => {
  if (!authContextLoading && !machineContextLoading && !locationContextLoading) {
    if (userToken !== null && machineId !== null && locationPermissionGiven) {
      setAppRequirementsMet(true);
    } else {
      setAppRequirementsMet(false);
    }
  }
}, [authContextLoading, machineContextLoading, locationContextLoading, userToken, machineId, locationPermissionGiven]);

const {position} = useNativeWatchPosition(appRequirementsMet);
```

KUVA 27. Sijainnin seurannan aloittaminen

4 POHDINTA

Tämän opinnäytetyötä koskevan projektin tuloksena onnistuttiin kehittämään vaatimuksensa täyttävä pieni mobiilisovellus, joka saatiin jo testattavaksi kehitystiimin ulkopuolisille henkilöille.

React Native osoittautui toimivaksi ohjelmistokehykseksi. Kehitystiimin verkkosivuohjelmoinnin kautta saadun Reactin osaamisen pystyi heti hyödyntämään React Nativessa. Suurin ero Reactin ja React Nativen välillä oli itse käyttöliittymän tyylittelytapa. Sekin oli nopeasti omaksuttavissa, kun ajatteli, että laaja-alaisen verkkokäyttöliittymän sijaan toteutettiin näkymää paljon pienempään näyttöön eli mobiililaitteen näyttöön. React Native mahdollisti mobiililaitteen natiivien rajapintojen käytön, eikä mikään projektin vaatimus jäänyt suorittamatta sen takia, että React Native olisi ollut jotenkin puutteellinen.

Reactin ja React Nativen suuren käyttäjäkunnan takia projektissa esiintyviin ongelmiin löytyi verkosta paljon apua ja valmiita ohjelmistokirjastoja. Sovelluksen kriittisin ja vaikein toiminto eli mobiililaitteen taustaseuranta täytyi kuitenkin toteuttaa itse, mutta senkin toteuttamiseen löytyi verkosta ohjeita. Syy miksei React Native sisältänyt valmista rajapintaa taustaseurantaan johtui mitä luultavimmin Android- ja iOS-käyttöjärjestelmien asettamista rajoituksista, joilla pyritään säästämään mobiililaitteen akkua ja kannustamaan erilaiseen optimoidumpaan vaihtoehtoon tiheän taustatoiminnon sijaan. Onneksi React Native kuitenkin mahdollisti omien natiivimoduulien teon, sillä tiheä taustaseuranta oli välttämätön ominaisuus projektillemme.

Oman sijainnin taustaseurantanatiivimoduulin toteutusta helpotti se, että sovelluksen oli määrä toimia vain Android-käyttöjärjestelmällä. React Native olisi kuitenkin mahdollistanut sovelluksen täyden toimimisen myös iOS-käyttöjärjestelmäisillä mobiililaitteilla vähäisin yhteensopivuus muokkauksin ja oman iOS-käyttöjärjestelmällä toimivan sijainnin taustaseurantanatiivimoduulin avulla.

Tämä sovellus oli kehitystiimin ensimmäinen projekti, jossa käytettiin React hookkeja. Hookit todettiin toimiviksi ja oppia niistä kertyi niin paljon, että niiden käyttö

muodostui sujuvaksi ja muillekin suositeltavaksi. Projektin aikana huomattiin, että monet viralliset Reactin ohjeet ja kolmannen osapuolen ohjelmistokirjastot sisälsivät myös hookein toteutettuja esimerkkejä, mikä on merkki hookkien nousevasta suosiosta.

Reactin virallinen tilanhallintatoteutus eli React konteksti -rajapinta osoittautui myös toimivaksi ja React hookit tukivat kontekstien käyttöä hyvin. Hookit olivat opinnäytetyön projektia tehtäessä vielä suhteellisen uusi lisäys Reactiin, joten kaikki tilanhallintakirjastot eivät välttämättä vielä mahdollistaneet hookkien käyttöä niin hyvin kuin olisivat voineet. Tämän takia Reactin virallisen tilanhallintarajapinnan käyttö oli turvallinen ratkaisu.

LÄHTEET

Android. Manifest.permission. Luettu 16.4.2020. https://developer.android.com/reference/android/Manifest.permission

Android. Permissions overview. Luettu 16.4.2020. https://developer.android.com/guide/topics/permissions/overview

Edelphi. Ohejlmistokehykset. Luettu 26.4.2020. https://www.edelphi.fi/ohjelmistokehykset/

Geeks for Geeks. Software Framework vs Library. Luettu 26.4.2020. https://www.geeksforgeeks.org/software-framework-vs-library/

GitHub. React Native Paper. Luettu 19.4.2020. https://callstack.github.io/react-native-paper/

GitHub. The React Native Community. Luettu 17.4.2020. https://github.com/react-native-community/.github

Google. Android. Luettu 13.4.2020. https://developers.google.com/android

Marbles. Frontend Developer. Luettu 26.4.2020. https://www.marbles.fi/digiammatit/arkkitehdit-ja-kehittajat/frontend-developer

Marbles. Web Developer ja Backend Developer. Luettu 26.4.2020. https://www.marbles.fi/digiammatit/arkkitehdit-ja-kehittajat/web-developer-backend-developer

Material Design. Introduction. Luettu 17.4.2020. https://material.io/design/introduction

Nielsen Norman Group. Mobile: Native Apps, Web Apps, and Hybrid Apps. Luettu 26.4.2020. https://www.nngroup.com/articles/mobile-native-apps/
React Native. AsyncStorage. Luettu 17.4.2020. https://reactnative.dev/docs/asyncstorage

React Native. Luettu 13.4.2020. https://reactnative.dev/

React Native. Native Modules. Luettu 26.4.2020. https://reactnative.dev/docs/native-modules-android

React Native. Native Modules. Luettu 7.5.2020. https://reactnative.dev/docs/native-modules-android

ReactJs. Components and Props. Luettu 10.4.2020. https://reactjs.org/docs/components-and-props.html

ReactJs. Context. Luettu 11.4.2020. https://reactjs.org/docs/context.html

ReactJs. Introducing Hooks. Luettu 11.4.2020. https://reactjs.org/docs/hooks-intro.html

ReactJs. Luettu 10.4.2020. https://reactjs.org/

ReactJs. React v16.3.0: New lifecycles and context API. Luettu 13.4.2020. https://reactjs.org/blog/2018/03/29/react-v-16-3.html

ReactJs. React v16.8: The One With Hooks. Luettu 11.4.2020. https://reactjs.org/blog/2019/02/06/react-v16.8.0.html

ReactJs. Rules of Hooks. Luettu 5.5.2020. https://reactjs.org/docs/hooks-rules.html

ReactJs. State and Lifecycle. Luettu 10.4.2020. https://reactjs.org/docs/state-and-lifecycle.html

ReactJs. Using the Effect Hook. Luettu 11.4.2020. https://reactjs.org/docs/hooks-effect.html

Robin Wieruch. React Function Components. Luettu 11.4.2020. https://www.robinwieruch.de/react-function-component

Stackon Technologies. API 23 – Normal and Dangerous Permissions in Android Marshmallow. Luettu 16.4.2020. http://stackontechnologies.com/api-23-normal-and-dangerous-permissions-in-android-marshmallow/

TomTom. Mikä on GPS? Luettu 26.4.2020. http://fi.support.tomtom.com/app/answers/detail/a_id/9138/~/mik%C3%A4-on-gps%3F

Valjas. Mitä integraatio, rajapinta ja api tarkoittavat? Luettu 26.4.2020. https://www.valjas.fi/mita-integraatio-rajapinta-ja-api-tarkoittavat/

Webopedia. What is Wi-Fi (Wireless?). Luettu 26.4.2020. https://www.webopedia.com/TERM/W/Wi_Fi.html